ТРИБУНА
МОЛОДЫХ УЧЕНЫХ
БИТ

С. С. Агафьин, А. А. Краснопевцев, П. В. Смирнов

ОБНАРУЖЕНИЕ НЕДОКУМЕНТИРОВАННЫХ ВОЗМОЖНОСТЕЙ ПРИЛОЖЕНИЙ

Актуальность проблемы

Основным носителем ключевой информации для криптографических приложений являются устройства на базе платформы JavaCard [1].

Существует множество методик анализа стойкости данных устройств [2–4], однако они не учитывают возможности внесения злоумышленником программных закладок (**недокументированных возможностей – НДВ**) еще на этапе разработки программного обеспечения.

Более того, к JavaCard в общем случае неприменимы классические методики защиты информации. Во-первых, это связано с тем, что создатели апплета могут не иметь доступа к разработке исполняемой среды (операционной системы) и, соответственно, ее интерфейсы чаще всего не удовлетворяют их потребностям (например, в силу ограниченной функциональности). Во-вторых, при использовании даже части функциональности интерфейсов операционной системы, являющихся расширением классического JavaCard, становится невозможной эмуляция приложений для данных платформ на многопоточных системах, что приводит к затруднению при проведении динамического анализа.

В настоящей статье предлагается методика, которая позволит определить наличие в апплете JavaCard недокументированных возможностей, использование которых может привести к нарушению конфиденциальности ключевой информации либо ее части.

Постановка задачи

Устройство на базе JavaCard может быть представлено в виде автомата Мили $(RAPDU, IS, CAPDU, \vartheta, \rho)$, где $RAPDU$ — множество стимулов, передаваемых на ключевой носитель; $CAPDU$ — множество реакций носителя на данные стимулы, IS — множество внутренних состояний; $\vartheta : RAPDU \times S \to CAPDU$; $\rho: RAPDU \times S \to S$.

Перед проверкой на наличие НДВ необходимо убедиться, что документированные функции, использующие ключевую информацию, являются корректными. В случае положительного результата тестов на корректность следует выделить множество $CI \subseteq RAPDU$ стимулов, при обработке которых может использоваться ключевая информация.

Таким образом, задача поиска недокументированных возможностей, способных привести к нарушению конфиденциальности ключевой информации, может быть сведена к задаче поиска таких стимулов $NDst \nsubseteq CI$, для которых существует полиномиальный алгоритм, способный по стимулу и соответствующей ему реакции $NDreac$ восстановить часть ключевой информации.

Методика обнаружения НДВ

Для удобства верификации с помощью доверенных средств проведем компиляцию исходного кода в промежуточное представление в виде CAP-файла, который затем будет интерпретироваться виртуальной машиной JavaCard [1].

По промежуточному представлению построим граф потока управления. В нем можно выделить три особых вида вершин: ST — точки входа в процесс исполнения текущей APDU-команды (по стандарту ISO 7816 ими могут являться только функции *select* и *process*); F — множество вершин, которые характеризуют успешное завершение обработки текущей APDU-команды; E — множество вершин, характеризующих неуспешное завершение текущей APDU-команды (генерация исключения).

В построенном графе методом экспертной оценки необходимо выделить переменную, которая хранит ключевую информацию. Соответственно, дальнейшая задача сводится к тому, чтобы определить, что к данной переменной осуществляется доступ только в рамках обработки стимулов, принадлежащих множеству CI.

Виртуальная машина JCVM, осуществляющая выполнение байт-кода рассматриваемого приложения для платформы JavaCard, может быть представлена в виде автомата (B,S,σ), где B — множество инструкций промежуточного представления JavaCard, используемых в данном приложении, $S = (Sst \times Svar)$ — множество внутренних состояний автомата, $Svar$ — множество состояний переменных, определенных для текущего приложения, Sst — множество состояний области памяти, куда помещаются аргументы функций перед их исполнением, σ — функция $(B \times S \to S)$.

Множество $Svar$ является вектором, i-я координата которого хранит значение переменной с индексом i. Множество Sst организовано по принципу стека и хранит номера переменных, значение которые будут использоваться при выполнении инструкции промежуточного представления.

Определим вектор $PL = (it_1, \dots, it_p)$, где $it_i = 1 \Leftrightarrow Svar_j[it_p] \in T$, где T — множество переменных, которые могут хранить в себе часть ключевой информации в состоянии S_i, а $it_i \in \{0,1\}\, \forall i \in \{1, \dots, p\}$. Вес данного двоичного вектора на первом шаге методики равен единице.

Введем функцию

$$\delta_{ij}: (S_i, S_j) \to (k_1, \dots, k_p): k_i = 1 \Leftrightarrow Svar_i[k_t] \neq Svar_j[k_t]\ t \in \{1, \dots, p\},$$

где $S_i, S_j \in S$, $k_i \in \{0,1\}, \forall i \in \{1, \dots, p\}$, а p — количество переменных, используемых в этом приложении. Данная функция указывает, какие переменные изменили свое значение при переходе автомата из состояния S_i в состояние S_j.

Тогда, последовательно применяя функцию δ для всех состояний S_i и S_j:

$\sigma(b_k, S_i) = S_j$, где $b_k \in B$, получим, что $PL = PL \vee \delta(S_i, S_j)$, если $\exists i \in : S_i st[l] \in T$, где $S_i st[l]$ — i-я переменная, хранимая на стеке виртуальной машины в состоянии S_i.

Имея перечень переменных, потенциально содержащих некоторую часть ключевой информации, можно далее найти в графе потока управления все пути, в которых значения этих переменных попадают в вершины из множеств E и F. Если множество стимулов, приводящих к исполнению промежуточного представления по данным путям, не совпадает с множеством CI, то программное обеспечение считается потенциально небезопасным и должно быть дополнительно исследовано экспертами либо направлено на доработку.

Заключение

Предлагаемая методика позволяет обнаружить недокументированные возможности программного обеспечения на базе платформы JavaCard и не использует допущения о возможностях разработчика.

Основным недостатком данной методики является высокая вероятность ошибки первого рода, связанная с тем, что локальные переменные, хранящие часть ключа, могут затем использоваться для хранения другой информации в рамках обработки одной APDU-команды. Для исключения таких ошибок предлагается ряд мер, которые будут подробно рассмотрены в дальнейших работах.

СПИСОК ЛИТЕРАТУРЫ:

1. Java Card Classic Platform Specification 3.0.4. URL: http://www.oracle.com/technetwork/java/ (дата обращения: 01.09.2013).
2. *Mostowski W., Poll E.* Malicious Code on Java Card Smartcards: Attacks and Countermeasures // LNCS. 2008. Vol. 5189. P. 1–16.
3. *Haneberg D., Grandy H., Reif W., Schellhorn G.* Verifying Smart Card Applications: An ASM Approach // LNCS. 2007. Vol. 4591. P. 313–332.
4. *Aussel J.* Smart Cards and Digital Security // MMM-ACNS. 2007. CCIS 1. P. 42–56.